VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Model autonomního řízení v simulátoru V-REP

Model of Autonomous Car Control in V-REP Simulator

2019 Michal Vašut

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Michal Vašut**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Model autonomního řízení v simulátoru V-REP
Model of Autonomous Car Control in V-REP Simulator

Jazyk vypracování: čeština

Zásady pro vypracování:

Cílem práce je vytvořit simulátor autonomního řízení modelu auta v prostředí V-REP (Virtual Robotic Experimental Platform). Prvním úkolem bude najít či sestavit model auta v měřítku odpovídajícím reálnému modelu - auto se dvěma motory a natáčením kol a řádkovou kamerou. Dále pak vytvořit základní stavební prvky dráhy - rovina, zatáčka, křižovatka, kopec, vlny - a připravit program, ve kterém si bude možno libovolnou dráhu sestavit. Pomocí API rozhraní simulátoru V-REP ověřit funkčnost celého modelu i kamery.

1. Seznamte se s prostředím V-REP, jeho ovládáním a tvorbou simulačního modelu.
2. Vytvořte dynamický model automobilu Alamak ve skutečném měřítku.
3. Vytvořte pro dynamický model automobilu jeho vizuální 3D obálku.
4. Vytvořte model jednotlivých dílů dráhy.
5. Napište potřebné programové rozhraní pro řízení auta s pomocí V-REP API v jazyce C/C++.
6. Funkčnost svého řešení ověřte na různých tvarech dráhy. Vyhodnoťte dosažené výsledky.

Seznam doporučené odborné literatury:

[1] V-REP - Virtual Robotics Experimentation Platform: http://www.coppeliarobotics.com/
[2] OpenSCAD, 3D modelování: https://www.openscad.org
[3] Blender, Open Source 3D creation: https://www.blender.org/

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Petr Olivka, Ph.D.**

Datum zadání: 01.09.2018
Datum odevzdání: 30.04.2019

doc. Ing. Jan Platoš, Ph.D.
vedoucí katedry

prof. Ing. Pavel Brandštetter, CSc.
děkan fakulty

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 29. dubna 2019 .....

Tímto bych rád poděkoval vedoucímu mé bakalářské práce Ing. Petru Olivkovi, Ph.D. za veškerou pomoc, cenné rady a připomínky.

Abstrakt

Tato bakalářská práce se zabývá tvorbou simulátoru pro autonomní řízení v prostředí V-REP. V rámci této práce byl vytvořen dynamický model auta v programu V-REP podle reálného modelu auta Alamak. Pro dynamický model vytvořeného auta byl také vytvořen vizuální vzhled podle reálného modelu Alamak. Na tento model byla umístěna řádková kamera a bylo vytvořeno rozhraní pro ovládání tohoto modelu auta z externí aplikace za pomocí V-REP API v jazyce C/C++. Pro testování ovládání byla vytvořena aplikace na ovládání auta pomocí gamepadu. Na toto ovládání byla navázána už existující aplikace pro autonomní ovládání reálného modelu. Pro testování autonomního řízení bylo potřeba vytvořit dráhu. Pro sestavení dráhy byly vymodelovány jednotlivé díly dráhy. Následně byl vytvořen skript, který ze zadaného řetězce znaků vygeneruje příslušnou dráhu.

Klíčová slova: V-REP, NXP Cup, Alamak, simulátor, Ackermannův podvozek, autonomní řízení

Abstract

This bachelor thesis deals with creation of simulator for autonomous car control in V-REP environment. The dynamic car model have been created in the V-REP simulator according to a real car model Alamak as a main part of this thesis. The visual appearance based on the real car model Alamak have been created for particular dynamic model. The line camera have been added to the car model and interface have been created to control this car model from an external application using V-REP API in C/C++. For testing purposes an application have been created to control the car model using a gamepad. An existing application for autonomous driving of the real car model have been connected to this interface. Track have been needed for test autonomous driving. Several track elements have been modeled for building track. Subsequently a script have been built which generates an appropriate track from specified string of characters.

Key Words: V-REP, NXP Cup, Alamak, simulator, Ackermann steering, autonomous driving

Obsah

Seznam použitých zkratek a symbolů

V-REP	–	Virtual Robot Experimentation Platform
API	–	Application Programming Interface
XML	–	Extensible Markup Language
LED	–	Light-Emitting Diode

Seznam obrázků

Seznam výpisů zdrojového kódu

1 Úvod

Cílem této bakalářské práce je vytvořit simulátor pro testování autonomního řízení v programu V-REP (Virtual Robot Experimentation Platform). V rámci toho je nutné se seznámit s prostředím simulátoru V-REP a následně v něm vytvořit dynamický model odpovídající reálnému modelu auta Alamak.

Tento model se používá při soutěži NXP Cup, kterou pořádá společnost NXP pro autonomně řízené modely. Tyto modely jsou řízeny podle dat z řádkové kamery. Dále je potřeba na virtuální model auta umístit řádkovou kameru a vytvořit rozhraní pro ovládání tohoto modelu pomocí V-REP API pro (C/C++).

Pro optimalizaci modelu auta bude vytvořeno ovládání pomocí gamepadu. Díky tomu je pak možné vytvořit rozhraní pro aplikaci na autonomní řízení modelu auta z dat řádkové kamery, které bylo vytvořeno jako bakalářská práce v minulých letech, pro ovládání reálného modelu auta. Ovládání je postaveno na rozpoznávání okrajových čar dráhy z řádkové kamery.

Pro testování autonomního řízení bude potřeba vytvořit také závodní dráhu. Proto budou vytvořeny jednotlivé prvky dráhy podle oficiálních podmínek pro NXP Cup a pro sestavení dráhy bude vytvořen skript, který sestaví dráhu, podle postupně zadaných prvků. Pro dynamický model bude také vytvořen vizuální vzhled odpovídající modelu Alamak.

Tento simulátor pro autonomní řízení umožní testování algoritmů pro autonomní ovládání auta. To by mohlo usnadnit práci při vývoji autonomního řízení s přesouváním ovládacího programu na reálný model auta. Bude také možné testovat autonomní řízení v různých situacích, protože v simulátoru je možné snadno měnit tvar dráhy.

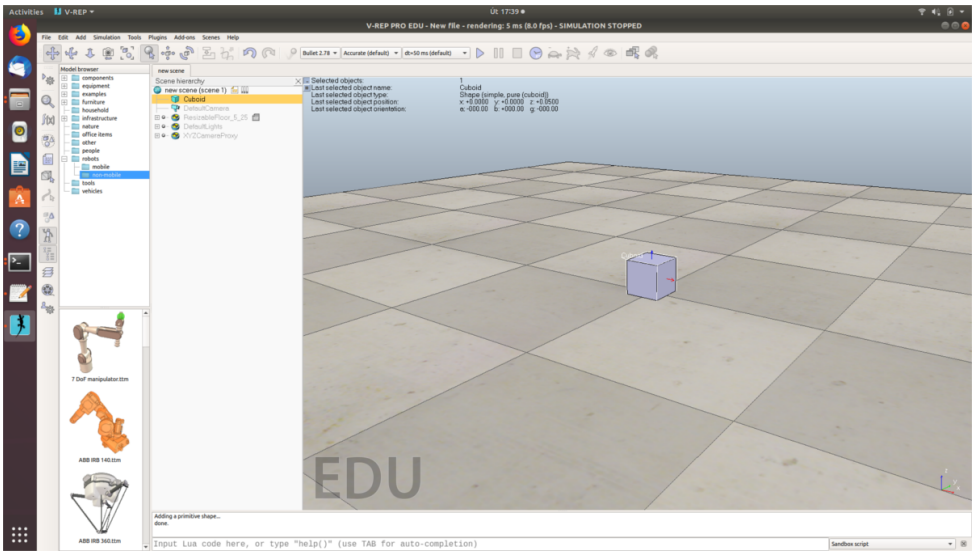
2 Popis prostředí V-REP

Virtual Robot Experimentation Platform [1] je simulátor pro testování robotických zařízení obsahující fyzikální engine. V tomto programu lze jednoduše navrhnout a otestovat virtuální prototyp robotického zařízení, bez potřeby jej fyzicky stavět. To může ušetřit náklady a čas spojený s konstruováním modelu. Zároveň je možné model bezpečně testovat bez hrozby poškození robota, nebo v podmínkách, které se špatně simulují ve skutečném prostředí.

2.1 Scéna

Hlavní částí simulátoru je scéna, která představuje pracovní prostor, do kterého je možné přidávat objekty a manipulovat s nimi. Do scény je možné přidávat různé typy objektů, jako jsou například tělesa, kamery, světla, klouby, senzory, grafy a další.

Dalším důležitým prvkem grafického prostředí je hierarchie scény. V ní jsou jsou zobrazovány všechny objekty, které jsou ve scéně. V tomto hierarchickém uspořádání je možné upravovat uspořádání nadřazenosti a podřízenosti jednotlivých objektů a dostávat se k jejich nastavení, nebo s nimi asociovat skripty. Jak vypadá scéna je možné vidět na obrázku 1, kde zabírá největší část prostředí V-REP. Hned nalevo od ní je hierarchie scény.



Obrázek 1: Prostředí programu V-REP

2.2 Simulace

Simulace je proces, kdy jsou na všechny objekty ve scéně aplikovány fyzikální zákony a pak je vidět jejich simulované chování. Simulace se ovládá na horním panelu, kde je možné zvolit start simulace, stop simulace, rychlost simulace a po jakých časových krocích má být fyzikální model aplikován. To se může projevit na náročnosti výkonu. V případě, že počítač nebude stíhat zpracovávat výpočty v reálném čase, bude docházet k viditelnému zpomalování simulace.

Po ukončení simulace jsou všechny objekty vráceny na jejich původní pozici i s jejich původním nastavením. V simulaci lze ovlivňovat jen některé možnosti nastavení objektů, jako je jejich pozice, viditelnost a podobně. V-REP počítá s fyzikálním modelem a dává možnost objektům nastavovat detaily, jako je váha a tření. Na počítání fyzikálního modelu je používán fyzikální engine nebo také dynamický modul. To je součást programu starající se o výpočet fyzikálního modelu. V-REP podporuje dynamické moduly Bullet 2.78, Bullet 2.83, Open Dynamics Engine (ODE), Vortex Dynamics a Newton Dynamics. Pro tuto bakalářskou práci byl vybrán základní engine Bullet 2.78 a 2.83.

Ve skutečnosti není možné počítat skutečný fyzikální model. Dynamický modul se snaží simulovat fyzikální model v reálném čase, k čemuž je zapotřebí mnoho optimalizací, díky kterým je výpočet efektivnější. Každý engine, počítá fyzikální model trošičku odlišně a může k výpočtům používat i jiné parametry podle toho, k čemu má být engine využíván. Dost časté využívání bývá v počítačových hrách, kde není potřeba přesnost, ale rychlost výpočtů fyziky, a proto se chování může lišit od skutečnosti.

2.3 Modely

Základ dynamického modelu se staví převážně z těles jednoduchých tvarů, které jsou seskupeny, nebo spojeny pomocí kloubů. V-REP obsahuje tři typy kloubů, které se dají kombinovat:

- Revolute joint (otočný kloub).
- Prismatic joint (posuvný kloub - píst).
- Spherical joint (kulový kloub).

Tyto klouby mají několik režimů chování. Základní režim je torque/force mode. V tomto režimu je simulována normální dynamika, kdy se kloub chová podle působení síly a také je možné mu nastavit, aby se kloub choval jako motor. Otáčení motoru lze pak ovládat v závislosti na rychlosti otáčení a točivém momentu. Pokud chceme kloubu volně nastavovat jeho natočení, je potřeba, aby byl nastaven v pasivní režimu.

V-REP také obsahuje ukázky už fungujích robotů, ze kterých je možné se učit a ve kterých jsou příklady pro používání všech funkcí, které program poskytuje. Také obsahuje už vytvořené modely pro práci, jako jsou například podlahy, terény, postavy, domácí nábytek, posuvné pásy, kostky se znaky pro rozpoznávání obrazu, nebo modely skutečných senzorů, motorů, úchopných zařízení atd.

2.4 Programovací rozhraní

V-REP obsahuje rozsáhlé možnosti pro ovládání modelů a simulace prostřednictvím poskytovaných rozhraní. Pro ovládání uvnitř simulátoru slouží regular API. Toto rozhraní poskytuje například skripty, do kterých je možné psát kód. Skripty se píší v jazyce Lua a je možné z nich

volat funkce pro ovládání simulace a objektů v simulaci. Zároveň skripty obsahují předpřipravené systémové funkce, které jsou volané programem V-REP. Skripty se dělí na tři druhy podle použití.

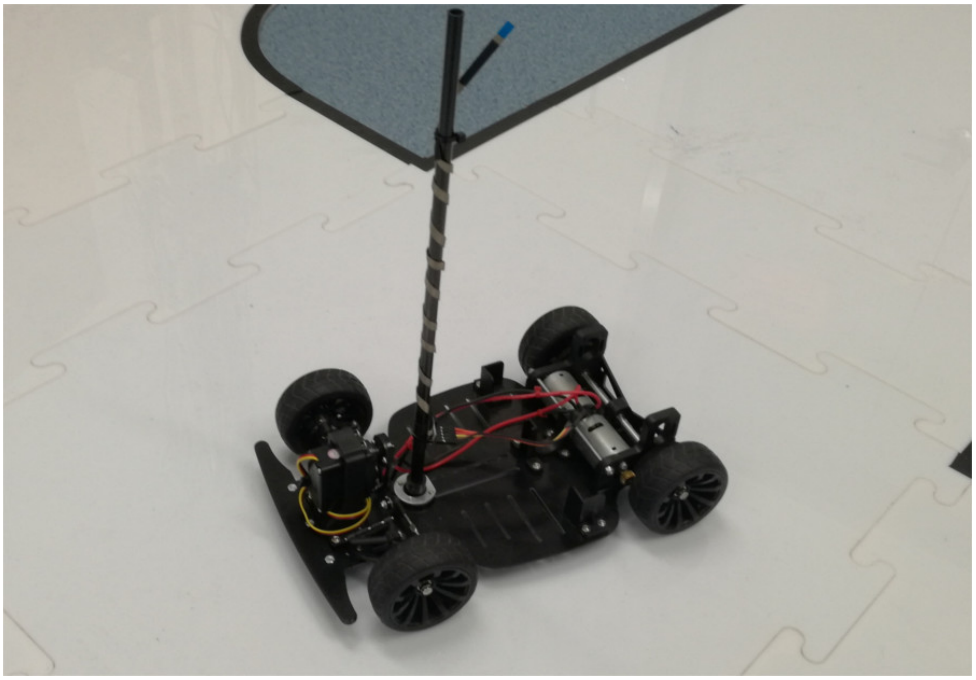
Main script je používaný pro nastavení scény a simulace. Tento skript není doporučováno upravovat nezkušeným uživatelům.

Další je *Child script*. Tyto skripty jsou vždy asociovány s některým z objektů ve scéně a jsou zpracovávány při běhu simulace. Tyto skripty se dělí ještě na běžící v odděleném vlákně nebo běžící jako součást simulace. Pokud je zvolen skript, který neběží v jiném vlákně, tak je vykonáván v každém kroku simulace. V těchto skriptech byl například napsán skript pro sestavení dráhy, nebo zajištění zatáčení kol podle Ackermannovy podmínky, jak bude popsáno později.

Posledním typem skriptu je *Customization script*. Ten je vykonáván, když je simulace vypnutá a využívá se třeba pro tvorbu nástrojů. Pro externí ovládání V-REP obsahuje rozhraní ROS, BlueZero nebo remote API. Pro tuto bakalářskou práci bylo důležité právě rozhraní remote API, které umožňuje ovládání simulátoru z aplikací napsaných pomocí jazyka Python, Java, Matlab nebo C/C++.

3 Tvorba modelu auta

Pro tuto bakalářskou práci je použit model Alamak [2] od firmy Landzo. Tento model se používá při soutěži NXP Cup pořádané firmou NXP. Model obsahuje dva motory na pohánění zadních kol a servo na zatáčení předních kol. Design přední nápravy je udělán tak, že při zatáčení dodržuje Ackermannovu podmínku, která je popsána v kapitole 3.1. Pro ovládání auta se používá řídící elektronika, kterou většinou tvoří nějaký mikropočítač. Vzhledem k tomu, že je v této bakalářské práci tvořen virtuální model auta, není tato ovládací elektronika potřeba. Reálný model auta je možné vidět na obrázku 2.



Obrázek 2: Model Alamak

3.1 Ackermannův podvozek

Je to mechanismus otáčení kol, díky kterému je splňovaná tzv. Ackermannova podmínka: střed natočení předních kol musí ležet na ose zadní nápravy [5]. To znamená, že se při průjezdu zatáčkou vnitřní kolo otočí více, než kolo vnější a díky tomu nevzniká smýkání kol. Tento mechanismus je použit u modelu Alamak a používá se i u normálních aut.

U reálného modelu auta je tvořen mechanicky, což by bylo složité modelovat i počítat pro simulaci, a proto je u virtuálního modelu natočení kol dopočítáváno z pomyslného středního kola. Znázornění Ackermannova zatáčení je na obrázku 3. Vzdálenost středu otáčení od osy auta se počítá podle natočení středního kola α_M :

$$R = \frac{L}{tg(\alpha_M)} \tag{1}$$

Pro výpočet natočení kola platí vzorec:

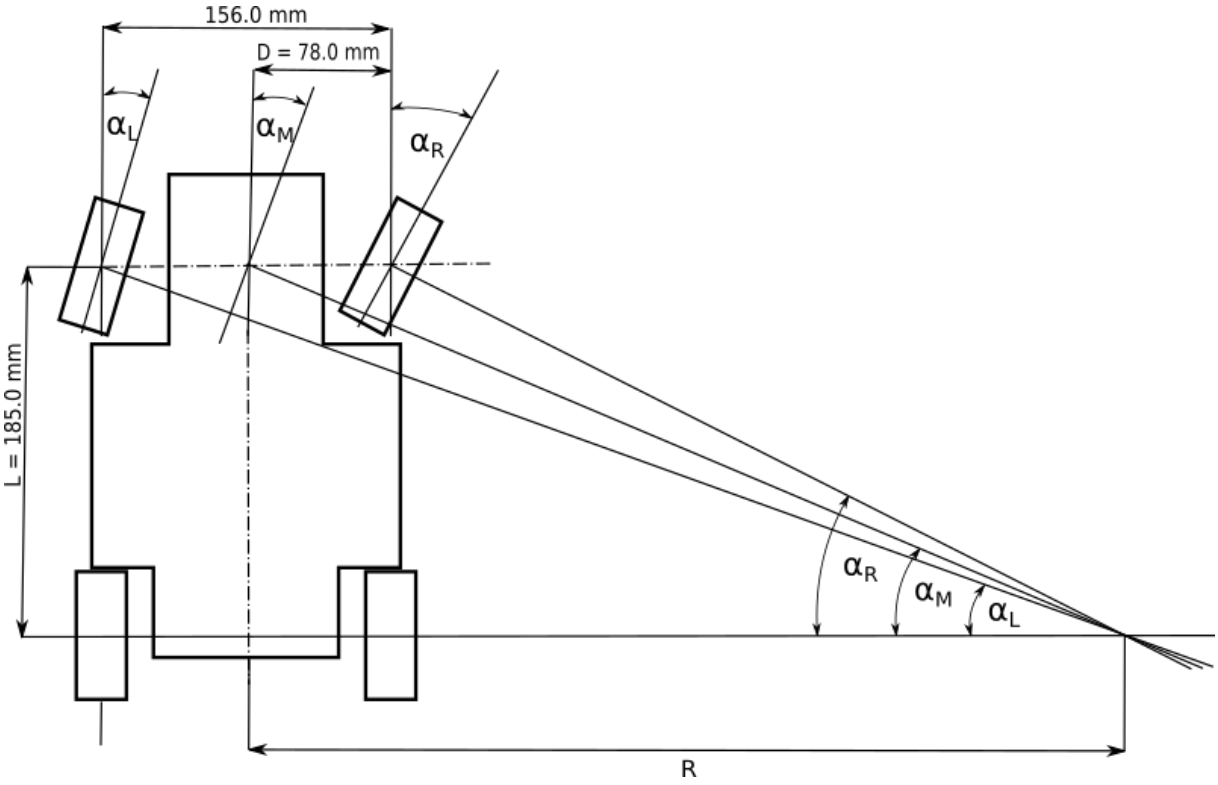
$$\alpha_L = arctg\Big(\frac{L}{R+D}\Big), \alpha_R = arctg\Big(\frac{L}{R-D}\Big) \tag{2}$$

Výsledný vzorec pro přepočet je tedy:

$$\alpha_L = arctg\left(\frac{L}{\frac{L}{tg(\alpha_M)}+D}\right), \alpha_R = arctg\left(\frac{L}{\frac{L}{tg(\alpha_M)}-D}\right) \tag{3}$$

kde jednotlivé proměnné představují:

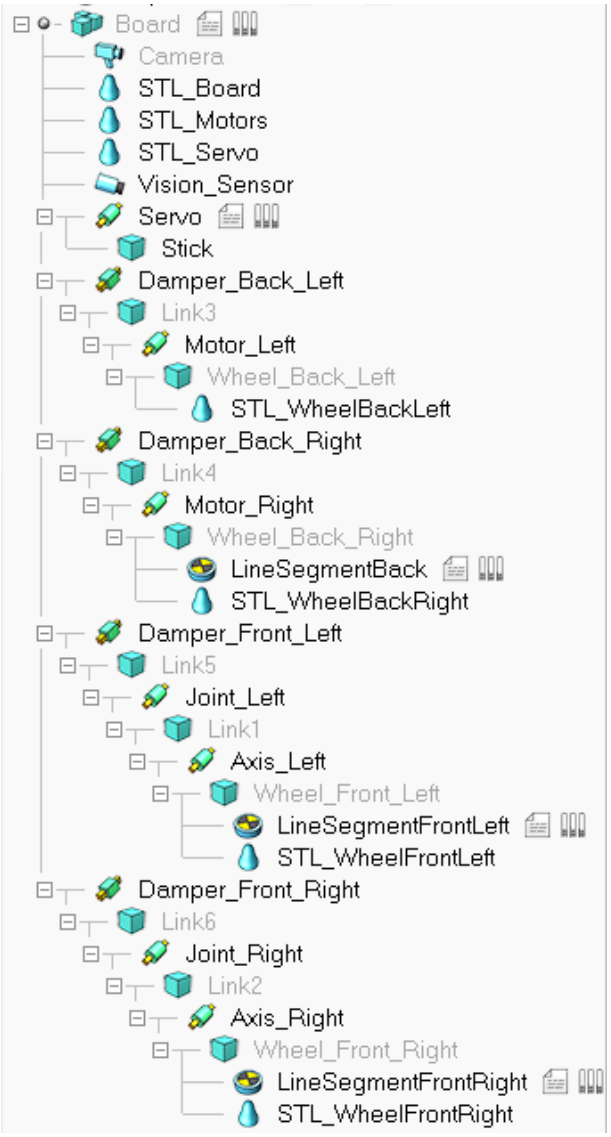
- α_M je úhel natočení středního kola.
- α_R je úhel natočení pravého kola.
- α_L je úhel natočení levého kola.
- L je vzdálenost přední a zadní nápravy 185 mm.
- D je polovina vzdálenosti mezi levým předním a pravým předním středem kol 78 mm.
- R je vzdálenost mezi středem otáčení a osou auta.



Obrázek 3: Geometrie Ackermannova podvozku

3.2 Tvorba dynamického modelu

Dynamický model je model auta v simulaci, který je součástí fyzikálního modelu, a proto by měl být co nejjednodušší. V simulátoru se modely sestavují hlavně z těles jednoduchých tvarů, které jsou vzájemně spojeny pomocí kloubů. Zbytek jsou senzory a objekty, které nejsou důležité pro dynamický model auta. Při dodržení správného spojení vzniká strom prvků. Při tvorbě modelu bylo potřeba si dávat pozor na to, aby dynamické části byly připojeny vždy jen na jiné dynamické části. Výsledný strom modelu auta je možné vidět na obrázku 4. Dynamický model a rozložení kloubů je možné vidět na obrázku 7.

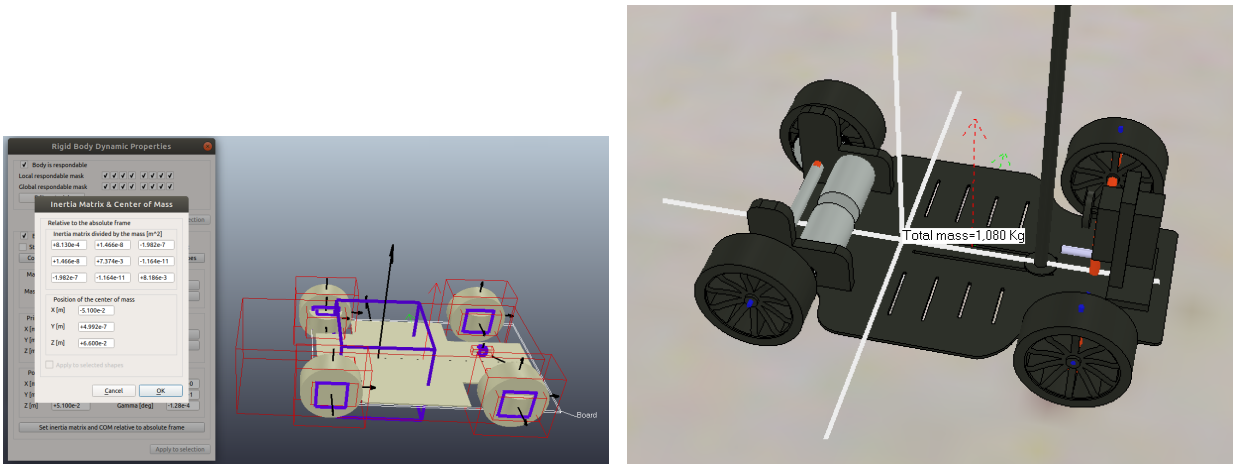


Obrázek 4: Hierarchie modelu auta

3.2.1 Podvozek

Podvozek je základní částí modelu, a proto byl zvolen jako nejvyšší prvek v hierarchické struktuře auta a je nastaven jako základ modelu. Pod tímto objektem jsou umístěné všechny objekty, které jsou součástí modelu auta. Aby model podvozku nebyl příliš náročný pro počítání kolizí v simulaci, je tvořen ze tří kvádrů, které jsou seskupeny do jednoho objektu.

Podvozek je dynamický, což znamená, že je součástí fyzikálního modelu a jsou u něj povoleny kolize. V nastavení dynamiky jde objektu nastavit jeho váhu. Celý skutečný model auta váží 1080 g a to včetně řídící elektroniky a baterie. Přední část modelu auta váží 460 g, zadní část modelu auta 620 g a kolo 30 g. Od podvozku byla odečtena váha kol a výsledná váha 900 g byla nastavená základní desce tak, aby těžiště modelu bylo v zadní části. Část váhy 60 g byla také rozpočítaná mezi objekty sloužící jako spoje mezi klouby na kolech. U těles jednoduchých tvarů lze upřesnit i střed těžiště objektu. Rozložení váhy modelu lze zobrazit ve scéně, jak je možné vidět na obrázku 5. Pomocí předpřipraveného nástroje je možné si zobrazit střed těžiště celého modelu, tzn. včetně kol.



(a) rozložení váhy modelu

(b) těžiště modelu

Obrázek 5: Znázornění váhy a těžiště auta

Na podvozek je přichycen také vision sensor, který reprezentuje řádkovou kameru. Vision sensor má rozlišení 128 x 1. Senzor není dynamický, a proto se pevně drží modelu. Výstup tohoto senzoru, lze také zobrazovat v okně *floating view*. Dynamická část podvozku je překrytá tělesy obecného tvaru představující vizuální vzhled modelu, které byly vymodelované v programu OpenSCAD. U podvozku to jsou objekty *STL_Board*,*STL_Servo* a *STL_Motors*. Jsou to tělesa, u kterých není povolena dynamika, aby nezatěžovaly simulaci. Těmhle tělesům je nastavena barva, aby co nejvíce připomínaly skutečný model. Objektům je možné nastavovat tzv. vrstvy viditelnosti. Ve scéně jsou vidět objekty, které mají povoleno zobrazování alespoň v jedné z viditelných vrstev. Proto, aby nebyly vidět zároveň tělesa dynamického modelu a vizuálního modelu, mají objekty nastaveny vrstvy viditelnosti různě. Díky tomu je možné ve scéně jed-

noduše přepínat mezi těmito náhledy. Dále je na rám připojena kamera, která se používá pro zobrazování scény v náhledu scény pokud je potřeba sledovat pohyb modelu auta.

3.2.2 Zadní náprava

Kola modelu auta jsou tvořeny válci. U těchto válců je povolená dynamika a kolize. Váha kola je nastavena na 30 g. Aby kola neprokluzovala při jízdě, je potřeba, aby vznikalo tření mezi dráhou a kolem. Tření u skutečného auta mezi pneumatikou a silnicí se pohybuje někde mezi 0.6 a 0.8 podle materiálu silnice [10]. Pro model bylo zvoleno tření 0.75. Výsledné tření mezi kolem a dráhou je v simulátoru počítáno jako násobek tření dráhy a tření kola. Aby bylo tření 0.75, bylo zvoleno tření dráhy 1 a tření kola 0.75. Tření se u těles jednoduchých tvarů nastavuje v úpravě materiálu. Na kola jsou také připevněné tělesa obecného tvaru představující vizuální vzhled kola. Kola jsou k základní desce připojené přes klouby. Zadní kola jsou k desce připevněné přes dva klouby. První kloub je *prismatic joint*, který představuje odpružení kol, nebo také měkkost pneumatik. Odpružení je nastaveno tak, aby se chovalo jako motor, který se snaží určitou silou tlačit vertikálně a tak napodobuje chování pružiny. Vzhledem k tomu, že není možné přesně změřit odpružení kol, tak byla síla motoru nastavena tak, aby odpružení při jízdě příliš neovlivňovalo chování modelu auta. Na tento kloub je připevněn kloub *revolute joint*, pro otáčení kola. Tento kloub představuje i pohon modelu a proto je nastaven jako motor. Tímto motorem je z externí aplikace ovládána rychlost modelu auta.

3.2.3 Přední náprava

Přední část modelu je složitější. Je potřeba simulovat i zatáčení kol podle Ackermannovy podmínky. Kola přední nápravy mají stejné vlastnosti, jako kola zadní nápravy, ale jsou k základní desce připojeny třemi klouby.

První je kloub *prismatic joint* představující odpružení kol. Na něj je připojen druhý kloub *revolute joint*, který umožňuje zatáčet. Tento *revolute joint* je nastaven v pasivním režimu, aby bylo možné kolům nastavit přesný směr natočení. A poslední *revolute joint* představuje osu otáčení kola. Ta se liší oproti zadním kolům v tom, že se nechová jako motor.

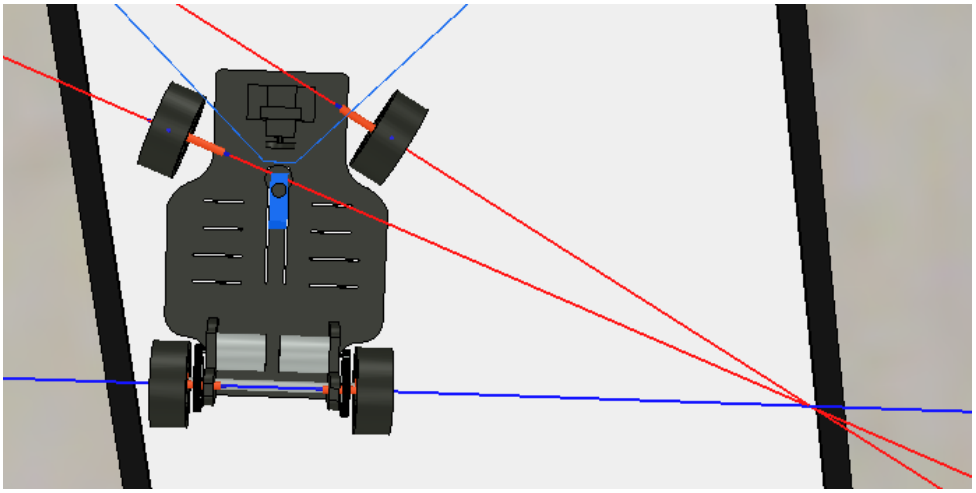
Pro zatáčení kol podle Ackermannovy podmínky, je na podvozek přidělán ještě jeden pomocný kloub *revolute joint*, který představuje zatáčení středního kola. Dalo by se říct, že je to servo reálného modelu. Toto servo není v pasivním režimu, jako klouby pro zatáčení krajních kol. Naopak se chová jako motor, kterému je nastaveno kontrolování natočení. Motor má nastaven cílový úhel natočení, kterého se snaží dosáhnout podle rychlosti a točivého momentu motoru. To je z toho důvodu, aby nebylo možné otočit koly instantně a aby to odpovídalo reálnému modelu. Toto servo má nastavené podobné hodnoty jako servo reálného modelu. Při napětí 4.8V servo reálného modelu dosahuje rychlosti $0.20s/60°$ to je rychlost 300° za sekundu a točivý moment je 7.5 kg na centimetr, což je asi 0.75 Nm. Směr natočení serva je ovládán z externí aplikace.

Pro přepočítávání natočení kol z tohoto pomyslného kola, byl s tímto servem asociován child script. V inicializaci tohoto skriptu jsou po startu simulace získané identifikátory vazeb ovládajících zatáčení kol, pomocí funkce `sim.getObjectHandle`. Ve funkci `sysCall_auction`, která se volá v každém kroku simulace, se získá natočení pomyslného středního kola pomocí funkce `sim.getJointPosition` a z něj se přepočítá natočení levého a pravého kola vzorcem 3, jak je možné vidět na výpise 1. Natočení kol se nastaví pomocí funkce `sim.setJointPosition`.

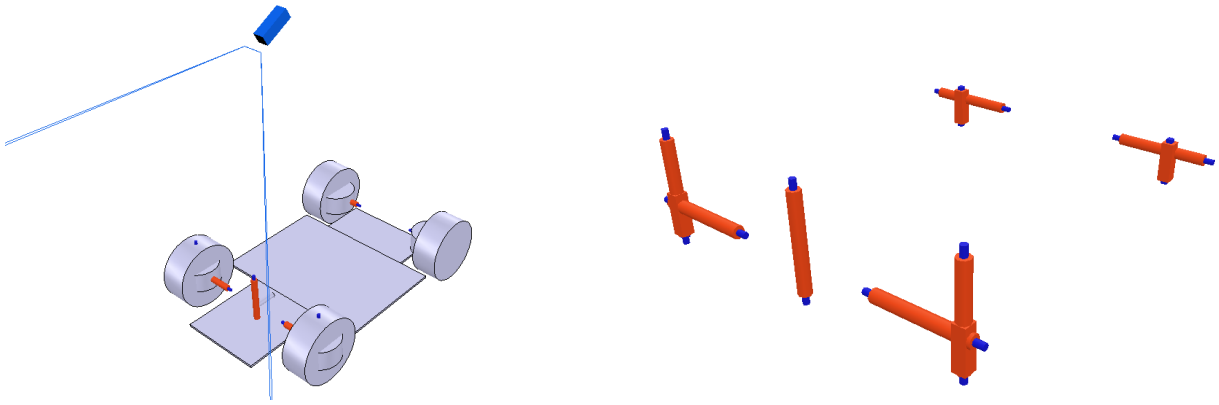
```
function sysCall_init()
   servo=sim.getObjectHandle('Servo')
   joint_L=sim.getObjectHandle('Joint_Left')
   joint_R=sim.getObjectHandle('Joint_Right')

   l = 0.185
   d = 0.078
end

function sysCall_actuation()
   angle_middle = sim.getJointPosition(servo)

   angle_right=math.atan(l/(-d+l/math.tan(angle_middle)))
   angle_left=math.atan(l/(d+l/math.tan(angle_middle)))

   sim.setJointPosition(joint_R, angle_right)
   sim.setJointPosition(joint_L, angle_left)
end
```

Výpis 1: Skript pro natáčení kol podle Ackermannovy podmínky

K ověření otáčení kol podle Ackermannovy podmínky byl vytvořen skript, který ve scéně vykresluje barevné čáry, které znázorňují osy kol. Proto byly na kola umístěny objekty dummy. Dummy je pomocný objekt, který se hodí na to, aby s ním mohl být asociován script. S těmito objekty byly asociovány skripty, které vykreslují barevné čáry ve směru osy z relativně k tomuto objektu. Tím, že jsou dummy objekty připojené na kola se jejich natočení mění stejně s koly a vykreslované čáry jsou tak shodné s osami kol. Jak je možné vidět na obrázku 6, prodloužené osy předních kol se při zatáčení kříží přesně na místě, kde je protíná i osa zadních kol, což potvrzuje funkčnost Ackermannova podvozku.



Obrázek 6: Osy kol auta



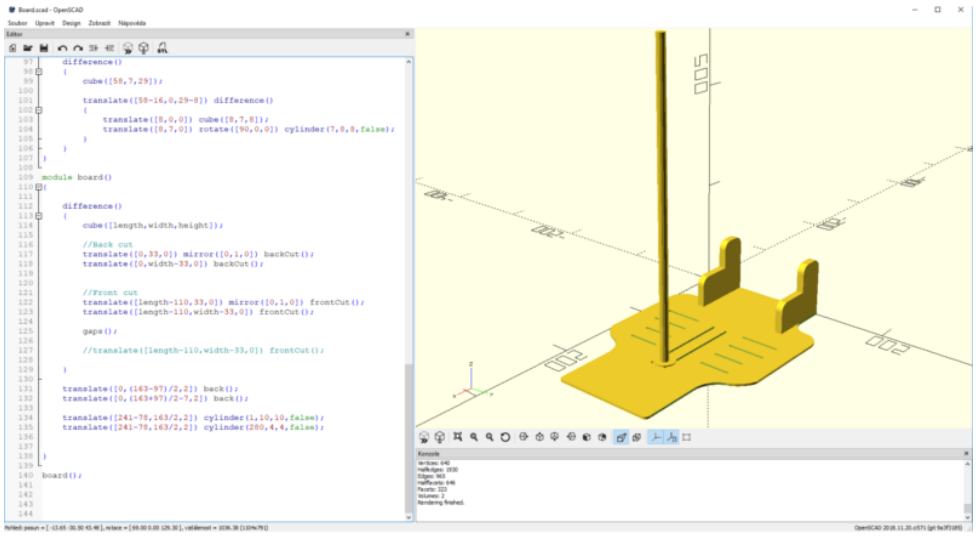
(a) dynamický model

(b) klouby modelu

Obrázek 7: Dynamický model

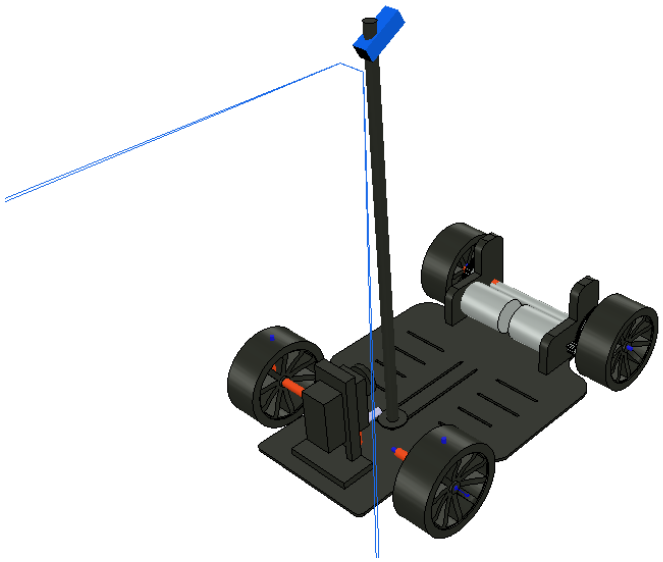
3.3 Modelování vizuálního vzhledu

Vizuální model auta byl vytvořen v programu pro 3D modelování OpenSCAD [3], ve kterém se model tvoří psaním skriptu. Skriptem lze vytvářet jednoduché tělesa a provádět s nimi základní operace, jako je například posun, sloučení dvou těles, rozdíl dvou těles a další. Díky tomuto se dají vytvářet potřebné modely. Výsledek skriptu se zobrazuje v grafickém náhledu, ve kterém je možné pohybovat kamerou, ale není možné přímo upravovat model. Ukázku prostředí programu OpenSCAD je možné vidět na obrázku 8, kde na levé straně je editor skriptu a na pravé straně je náhled modelu. Jedná se o vymodelovanou desku modelu auta.



Obrázek 8: Prostředí OpenSCAD

Těmito způsoby byly vytvořeny jednotlivé části auta, tzn. kolo, deska, servo a motory. Výsledné modely byly uložené do souboru STL, který je právě podporován simulátorem V-REP. Po importování do prostředí V-REP jsou tyto modely představovány objekty obecného tvaru. Tyto objekty překrývají v simulátoru dynamický model auta a bude u nich vypnuta dynamika. Důvod, proč nejsou tyto části použity jako součást dynamického modelu je ten, že počítání fyzikálního modelu s tělesem obecného tvaru je velice náročné a mohlo by významně zpomalovat simulaci. Vizuální vzhled auta je možné vidět na obrázku 9. Jednotlivé díly byly vymodelované podle naměřených hodnot, které jsou zakresleny do obrázku v příloze B. Jedná se jen o přibližný model auta a proto vizuální model neobsahuje některé funkční části modelu auta.



Obrázek 9: Vizuální vzhled virtuálního modelu

4 Ovládání auta

4.1 Ovladač

Pro optimalizaci ovládání auta při vývoji byl použit bezdrátový gamepad F710 od společnosti Logitech. Jedná se o klasický ovladač se dvěma joysticky, osmisměrným digitálním tlačítkem, akčními tlačítky A,B,X,Y, zadními analogovými a digitálními tlačítky a funkcí vibrace. Je možné také prohodit ovládání osmisměrového tlačítka a joysticku. Z ovladače byly využity oba joysticky a tlačítko RB. První joystick je používán na ovládání rychlosti a druhý joystick k ovládání směru jízdy auta. Tlačítko RB je použito pro restart vozidla na počáteční pozici. Ukázka ovládání na daném ovladači je vidět na obrázku 10.

Obrázek 10: Gamepad Logitech F710

Vývoj aplikace probíhal na systému linux a ten umožňuje přistupovat k připojeným zařízením, jako je gamepad podobně, jako k souborům.[9] Připojený ovladač se zobrazí v souborovém systému jako soubor */dev/input/js0*. Pro čtení hodnot gamepadu stačí otevřít soubor funkcí `open`. Čtení pak probíhá pomocí funkce `read`, do které je vložen file descriptor a odkaz na proměnnou správné struktury. Do této struktury jsou zapsána poslední přijatá data gamepadu. Struktura obsahuje čas, hodnotu, typ a číslo prvku. Levý i pravý joystick vrací hodnoty od -32767 do 32767 pro osu x a pro osu y.

4.2 Rozhraní remote API

Model auta je ovládán z externí aplikace napsané v jazyce C/C++. Pro ovládání simulace nabízí V-REP remote API. Remote API je rozhraní, které umožňuje kontrolovat simulaci z externí aplikace. Rozhraní komunikuje s programem V-REP přes síťové sockety, a proto je nutné, aby na straně simulátoru běžel remote API server.

Veškerou komunikaci mezi simulátorem V-REP a externí aplikací si řeší API rozhraní samo. Pro začátek komunikace je potřeba navázat spojení se serverem. Remote API nabízí pro ovládání podobné funkce jako vestavěné rozhraní regular API. Všechny funkce remote API mají jako parametr identifikátor klienta, ze kterého je funkce volána a parametr, který určuje režim, ve kterém pracují. Základní dělení režimů je na blocking a non-blocking. Blokující režim pozastaví běh programu, dokud nepřijdou data z programu V-REP. Zbývající režimy jsou různé způsoby, jakými se řeší čekání na data tak, aby nebyl blokován chod programu. Aplikace pro řízení auta jsou dělány tak, aby neblokovaly chod programu. Pro přeložení externí aplikace do spustitelného programu je potřeba připojit několik důležitých souborů remote API.

4.3 Ovládání

U modelu auta je ovládána rychlost a natočení předních kol.

4.3.1 Natočení předních kol

U reálného modelu auta bylo změřeno maximální možné natočení kola 30°. Z aplikace je ale ovládáno pomyslné střední kolo. Pokud by bylo natočeno na 30°, pak by vnitřní kolo auta bylo natočeno v úhlu přesahující 30°, kvůli implementování ackermannova podvozku. Je nutné spočítat maximální možné natočení středního kola proto, aby maximální natočení vnitřního kola, při průjezdu zatáčkou, bylo vždy maximálně 30°. Vzorec pro výpočet je stejný jako při přepočtu akcermannovy podmínky ve vzorci 3. Tedy:

$$\alpha_M = arctg\left(\frac{L}{\frac{L}{tg(30°)} - D}\right) \tag{4}$$

4.3.2 Ovládání rychlosti

U auta je rychlost ovládána pomocí točivého momentu. To vypadá tak, že je rychlost motoru nastavena na maximální povolenou hodnotu. Této hodnoty se pak motor snaží dosáhnout podle toho, jak velký má točivý moment. Čím vyšší má točivý moment, tím rychleji ji dosáhne. Maximální rychlost motoru byla omezena tak, aby rychlost modelu auta byla maximálně 1.2 m/s a rychlost nebyla zvyšována do nekonečna.

Pro získání odpovídajícího točivého momentu reálného modelu byl proveden pokus. Na kolo reálného modelu auta byla připevněna 90 cm dlouhá tyč tak, aby kolo bylo ve středu tyče. Motor byl pak spuštěn na maximální výkon a bylo testováno, jakou silou tyč působí na kuchyňskou

váhu. Při maximálním výkonu motoru tlačila tyč na váhu silou odpovídající hmotnosti 22 g. Znázornění pokusu je na obrázku 11. Pro výpočet výsledného točivého momentu je pak použitý vzorec. Výpočet výsledného točivého momentu:

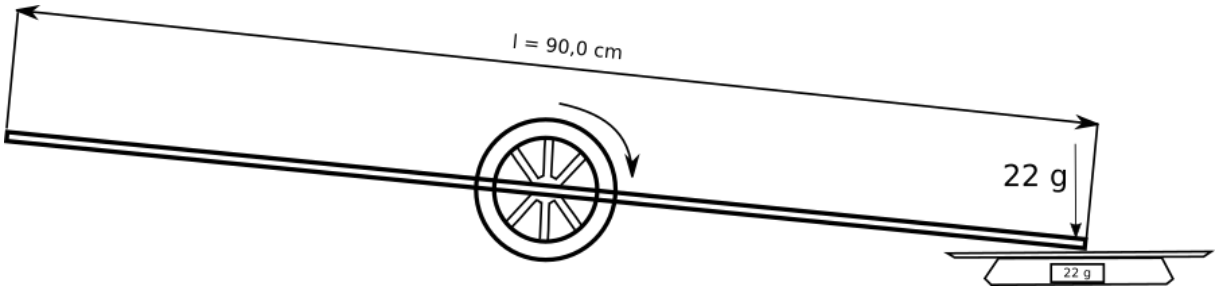
$$F = m \cdot g$$

$$M = r \cdot F$$

$$M_k = 0.22 \cdot 0.45 = 0.099\,Nm \tag{5}$$

kde jednotlivé proměnné představují:

- F je síla v N.
- m je váha v kg.
- g gravitační konstanta zaokrouhlována na 10.
- M,M_k je točivý moment.
- r je rameno působící síly l/2, 0.45 m.



Obrázek 11: Znázornění měření M_k motoru

Při takto vysokých hodnotách se auto stává neovladatelným a proto se M_k omezuje až pod polovinu maximálního možného M_k.

4.4 Remote API na straně simulátoru

Remote API server je spouštěn pomocí skriptu u objektu `RemoteAPI_Start` při startu simulace. Skript vybere volný síťový port, na kterém bude server naslouchat. Pak zkontroluje, zda obsahuje všechny potřebné soubory pro spuštění serveru. Jde hlavně o plugin remote API, který je součástí programu V-REP. Pokud vše projde, tak je server zapnut funkcí `sim.RemoteApi.start(port)`. Ze skriptu je možné spustit externí aplikaci pro ovládání auta pomocí funkce `sim.launchExecutable`.

4.5 Algoritmus na ovládání auta pomocí ovladače

Pro ovládání bylo definováno několik důležitých hodnot. První z nich je maximální natočení vnitřního kola při zatáčení. To je u reálného modelu Alamak 30°. Podle něj se pak vypočítá maximální natočení středního kola podle vzorce 4. Další hodnotou je rychlost otáčení zadního motoru. Ta je počítána z rychlosti auta zadáváné v metrech za sekundu. Z ní se pak podle obvodu kola spočítá rychlost otáčení kola za sekundu. Dále jsou staticky definovány hodnoty maximálního točivého momentu a počet kroků osy gamepadu. Všechny definované hodnoty je možné vidět ve výpise 2.

```
#define MAX_ANGLE_INNER_WHEEL_DEG 30
#define MAX_ANGLE_RAD atan(185/(78+(185/tan(MAX_ANGLE_INNER_WHEEL_DEG*M_PI/180)
    )))
#define MAX_SPEED_M_S 1.2
#define MAX_SPEED_DEG_S ((MAX_SPEED_M_S*1000)/(64*M_PI))*360
#define MAX_TORQUE 0.03

#define GAMEPAD_STEPS 32767
```

Výpis 2: Definování statických hodnot

Program přijímá jako argument port, na kterém naslouchá V-REP remote API server. Tento port je využit pro připojení k serveru pomocí funkce `simxStart`, která v případě, že je úspěšná, vrací hodnotu klienta. Tuto hodnotu, je důležité uložit, protože je následně používaná pro identifikaci ve všech následujících V-REP API funkcích. Pokud připojení proběhne úspěšně, tak jsou zaslány žádosti o identifikátory motorů, serva a řádkové kamery. Následně je dobré si určit kroky, po kterých se bude zvedat točivý moment a úhel natočení středního kola v závislosti na poloze joysticku. Joystick vrací do jednoho směru rozsah od 0 do 32767. Velikost kroku se počítá podle výrazů ve výpise 3.

```
float stepAngle= MAX_ANGLE_RAD/GAMEPAD_STEPS;
float stepTorque= MAX_TORQUE/GAMEPAD_STEPS;
```

Výpis 3: Definování velikosti kroků natočení a točivého momentu

Před zahájením cyklu je ještě nutné si nadeklarovat několik pomocných hodnot a zahájit čtení z gamepadu. Čtení gamepadu probíhá v odděleném vlákně proto, aby mohlo probíhat nezávisle. Toto vlákno je potřeba spustit a předat mu několik parametrů. Pro předávání parametrů byla vytvořena struktura. V této struktuře jsou ukazatelé na proměnné `osaX`, `osaY`, `restartButton`, `stopThread` a `joystick_fd`. File descriptor slouží pro uvedení, který gamepad se bude používat. Pomocná proměnná `stopThread` slouží k zastavení vlákna. Do zbývajících proměnných se ukládá

poslední hodnota přečtená gamepadem. V tomto vlákně je pouze cyklus, který provádí čtení z gamepadu. Podle čísla funkčního prvku se pozná, která páčka byla použita a z ní se přečte hodnota. Hodnota je uložena do proměnné `osaX` nebo `osaY` podle toho, která páčka změnila svou hodnotu. Právě v těchto proměnných jsou uloženy poslední hodnoty páček. V hlavním programu je v cyklu prováděn výpočet natočení serva, točivého momentu a rychlosti kol, podle hodnot ovladače. Vypočítané hodnoty jsou následně nastaveny motorům pomocí funkcí remote API. Viz výpis 4.

```
angle = osaY*stepAngle;
torque = osaX*stepTorque;
speed = osaX;

if (speed>3) speed=-MAX_SPEED_DEG_S; else
if (speed<-3) speed=MAX_SPEED_DEG_S; else
speed = 0;

if (torque<0) torque*=-1;
simxSetJointTargetVelocity(clientID,leftMotorHandle,speed,simx_opmode_oneshot);
simxSetJointForce(clientID,leftMotorHandle,torque,simx_opmode_oneshot);
simxSetJointTargetVelocity(clientID,rightMotorHandle,speed,simx_opmode_oneshot)
    ;
simxSetJointForce(clientID,rightMotorHandle,torque,simx_opmode_oneshot);


simxSetJointTargetPosition(clientID,servoHandle,angle,simx_opmode_oneshot);
```

Výpis 4: Nastavení rychlosti

4.6 Možnost restartu auta

Pro snadnější testování byla přidána funkcionalita pro restart auta. Protože je funkce složitější, byla implementovaná na straně simulátoru v *child scriptu* asociovaného s deskou modelu auta. Funkce je volána z klientské aplikace v případě, že je na gamepadu zmáčknuto tlačítko RB. Ukázka volání funkce `restart` z externí aplikace je ve výpise 5.

```
simxCallScriptFunction(clientID, "Board", sim_scripttype_childscript , "restart
    ", 0, NULL, 0, NULL, 0,NULL,0, NULL,0, NULL, 0, NULL, 0, NULL, 0, NULL,
    simx_opmode_blocking);
```

Výpis 5: Funkce pro zavolání funkce na serveru

Funkce pro restart auta je ukázána ve výpise 6. Skript v simulátoru při startu simulace uloží počáteční pozici a natočení modelu auta. V případě zavolání funkce restart z externí aplikace jsou uloženy všechny objekty, které jsou součástí modelu auta. Zároveň musí skript běžet buď v režimu, kdy neběží v odděleném vlákně, nebo krátkodobě nastavit, aby neběžel v odděleném vlákně. Důvodem je, že by se model mohl rozpadnout při přenosu kvůli aplikování působení fyzikálních sil. Následně je potřeba uložit konfiguraci stromu modelu. Pak projít všechny objekty modelu auta a u každého restartovat dynamiku objektu. To znamená restartovat všechny síly, které na něj působí a nastavit uložené hodnoty konfigurace zpátky. Pak je teprve možné bezpečně přesunout model na počáteční pozici.

```
function restart()

   handlers = sim.getObjectsInTree(board, sim.handle_all, 2)
   robotInitialConfig = sim.getConfigurationTree(board)
   --sim.setThreadAutomaticSwitch(false)

   for i=1,#handlers,1 do
      sim.resetDynamicObject(handlers[i])
      print(handlers[i])
   end
   sim.setConfigurationTree(robotInitialConfig)
   sim.setObjectPosition(board, -1, startPosition)
   sim.setObjectOrientation(board, -1, startOrientation)
   --sim.setThreadAutomaticSwitch(true)

   return {},{},{},''
end
```

Výpis 6: Restarování modelu auta na startovní pozici

4.7 Vykreslování řádkové kamery

Pro větší přehled o tom, jaké data aplikace z rozhraní dostává byl vytvořen kód, který vykresluje obraz z řádkové kamery do grafického okna.

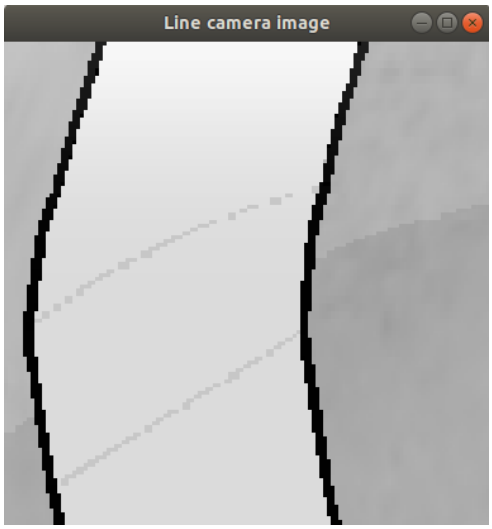
Na toto vykreslování byla použita knihovna OpenCV [4]. Ta umožňuje vytvořit nové okno, ve kterém je možné zobrazit obrázek. Čtení obrazu z kamery a vykreslování se provádí v hlavním cyklu, ale ke správnému fungování je potřeba připravit několik věcí. Je potřeba vytvořit nové okno a matici pro obrázek, do kterého se bude vykreslovat obraz. Do tohoto obrázku se zakresluje obraz z kamery v průběhu jízdy. Současný obraz z kamery je zakreslen na první řádek obrázku.

Všechny ostatní řádky jsou posunuty o jeden řádek dolů a tím vzniká krátký záznam z kamery. Tento kód se provádí v odděleném vlákně, aby nezpomaloval hlavní část programu. Pro toto vlákno je nutné nadefinovat atribut, aby se vlákno po proběhnutí správně ukončilo a nečekalo se na přečtení jeho návratové hodnoty. Čtení z řádkové kamery je doporučováno dělat v režimu *streaming*. To znamená, že se čtení neprovádí klasicky tak, že je zaslán požadavek o obraz na server, ale server automaticky posílá v každém kroku simulace nový obraz klientu automaticky. Toto čtení je potřeba zahájit tím, že zašleme žádost o obraz z kamery ve *streaming* režimu. V hlavním cyklu se pak provádí čtení z kamery v režimu *buffer*, což znamená, že se obraz čte z bufferu klienta. V bufferu je uložený poslední obraz, který ze serveru přišel. Tímto dochází k velkému zrychlení celého programu. Je ale zbytečné několikrát zpracovávat ten stejný obraz, proto je po zpracování tohoto obrazu smazán obraz z bufferu. Při čtení je pak udělána krátká čekací smyčka, dokud se v bufferu neobjeví nový obraz. Tím dochází i ke zpomalení zpracovávání asi na rychlost kroku simulace. Při čekání na nový obraz z kamery je také důležité kontrolovat, zda nedošlo k vypnutí simulace, aby nedošlo k zamrznutí aplikace. Po každém přečtení obrazu z řádkové kamery je spuštěno nové vlákno, které provádí posun řádků obrázku a zakreslování aktuálního záznamu. Zobrazení obrázku v okně je prováděno v hlavním cyklu programu. To se dělá s omezením v každém patnáctém cyklu proto, aby zbytečně nedocházelo k zatěžování. Na obrázku 12 je ukázka okna, ve kterém je vykreslovaný obraz řádkové kamery. Ve výpise 7 je upravená ukázka kódu, ve které je možné vidět nastavení a čtení obrazu z kamery a její vykreslování.

```
Mat image;

pthread_t drawing;
pthread_attr_t thread_atribut;
pthread_attr_init(&thread_atribut);
pthread_attr_setdetachstate(&thread_atribut, PTHREAD_CREATE_DETACHED);
simxUChar* image_camera;
int resolution[2];
simxGetVisionSensorImage(clientID,visionSensorHandle,resolution,&image_camera
    ,1,simx_opmode_streaming);

image = Mat::zeros(Size(128, 256), CV_8UC3);
int time_to_draw = 15;

namedWindow( "Line camera image", WINDOW_NORMAL );
resizeWindow("Line camera image",128*3,256*3);
while (simxGetConnectionId(clientID)!=-1)
{
```

```
    if (time_to_draw<0)
    {
          imshow( "Line camera image", image );
       waitKey(1);
       time_to_draw = 15;
    }
    time_to_draw--;


        int retVal;
        while((retVal=simxGetVisionSensorImage(clientID,visionSensorHandle,
            resolution,&image_camera,1,simx_opmode_buffer)!=simx_return_ok)&&(
            simxGetConnectionId(clientID)!=-1))
        {
           extApi_sleepMs(1);
        }

        if (retVal!=-1) pthread_create(&drawing, &thread_atribut, draw_thread,
            image_camera);

        simxGetVisionSensorImage(clientID,visionSensorHandle,resolution,&
            image_camera,1,simx_opmode_remove);
}
```

Výpis 7: Čtení a vykreslování obrazu kamery z externí aplikace

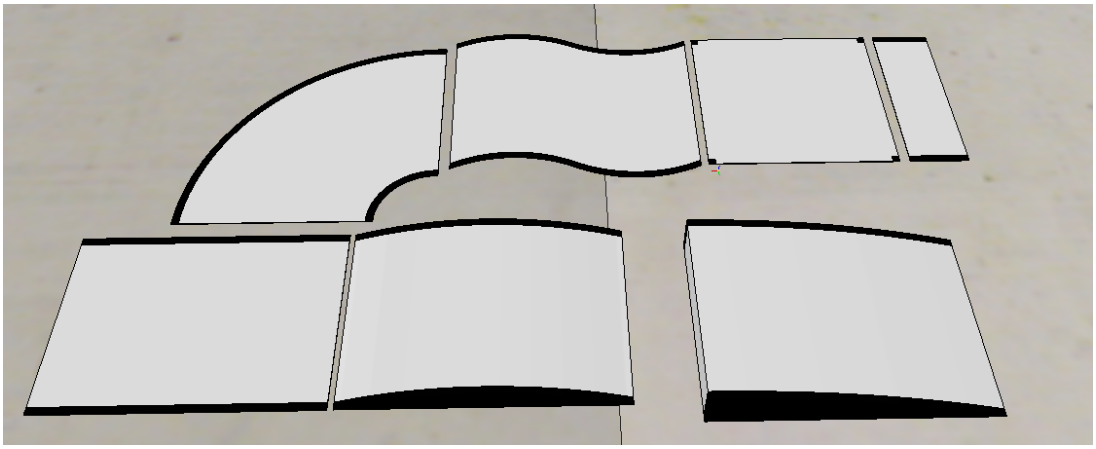


Obrázek 12: Vykreslování řádkové kamery pomocí OpenCV

5 Dráha

Testovací dráha se skládá z mnoha dílů různých tvarů. Jde o mírně upravené části, které jsou používány při soutěži NXP Cup. Protože je dráha používána pro testování autonomního řízení, založeného na rozpoznávání dat z kamery, je pro to přizpůsobená. Oficiální specifikace dráhy lze najít na stránkách pořadatele [8].

Jedná se o bílou dráhu s černými okraji, aby se dalo dobře rozpoznat, kudy má auto jet. Šířka dílu dráhy je 550 mm a z toho jsou 20 mm široké černé okraje. Výška dráhy je 2 mm. Mezi prvky dráhy patří rovina, levotočivá a pravotočivá zatáčka, šikana, křižovatka, krátká doplňovací rovina, malý kopec a velký kopec. Všechny tyto části jsou na obrázku 13. Kromě zatáčky a krátké roviny mají všechny díly délku 720 mm. Zatáčka 90° má vnitřní poloměr 170 mm. Velký kopec se skládá ze dvou dílů a má výšku 100 mm. Výška malého kopce je 60 mm. Šikana má poloměr jednotlivých zatočení 400 mm. Krátká rovina má délku 170 mm a používá se jako doplnění, když dráha nenavazuje přesně na sebe. Křižovatka je čtverec o šířce 550 mm. Právě na doplnění délky křižovatky je používána krátká rovina.



Obrázek 13: Prvky dráhy

5.1 Modelování dráhy

Modelování dráhy, stejně jako modelování auta, probíhalo v programu OpenSCAD. Některé části dráhy byly vymodelované už v minulých letech a ty bylo potřeba jenom trochu upravit.

Modely dráhy byly do programu V-REP importovány ve formátu STL, který nepodporuje ukládání barev. Všechny modely byly proto dobarvovány přímo v programu V-REP. Musely být modelovány tak, aby byly odděleny různě barevné části. U všech dílů dráhy bylo nutné posunout černý okraj od středu dráhy a po importu a obarvení je zase posunout zpátky a sloučit je do jednoho objektu.

5.2 Generování dráhy

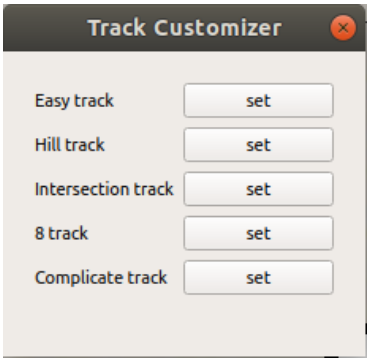
Generování dráhy se provádí pomocí skriptu asociovaného s dummy objektem *MapGenerator*, pod který byly přiřazeny i všechny díly dráhy. Dráha je generována podle zadaného řetězce písmen, kdy každé písmeno představuje právě jednu část dráhy. Přímo ve skriptu lze přepsat řetězec a tím i tvar dráhy, nebo využít možnost zvolit jednu z předpřipravených tratí. Proto je objekt dummy asociován se dvěma skripty.

První je child script, který se provádí při běhu simulace a druhý je *customization script*, který je naopak vykonáván, když simulace neběží. *Customization script* se stará o to, aby bylo možné jednoduše přepínat mezi předpřipravenými tratěmi. Tento skript zajišťuje, že při vybrání dummy objektu *MapGenerator*, se otevře dialogové okno s nabídkou tratí. Tyto vestavěné skripty obsahují předdefinované funkce, které jsou volané programem V-REP. Pro inicializaci je funkce `sysCall_init` volána po každé změně skriptu, nebo po načtení nové scény. V této funkci se uloží identifikátor pro objekt *MapGenerator* pomocí funkce `sim.getObjectAssociatedWithScript` a je definována proměnná identifikující dráhu `trackID`. Další funkcí je `sysCall_nonSimulation()`, která se volá opakovaně, pokud zrovna neběží simulace. V této funkci je kontrolováno, jestli vybraný objekt je objekt *MapGenerator*. Pokud vybraný objekt je *MapGenerator*, tak se volá funkce `showDlg()`, pokud není, tak je volána funkce `hideDlg()`. Ve funkci `showDlg` je kontrolováno, zda už není okno otevřené, a pokud není, tak vytváří nové. V-REP dává možnost jednoduše vytvořit nový prvek uživatelského rozhraní pomocí jazyka XML. V něm lze jednoduše definovat, jak má okno vypadat a jdou do něj přidávat jednotlivé prvky uživatelského rozhraní, jako je tlačítko, combobox, checkbox, progressbar, radiobutton, obrázek, text, záložka a další. U prvků je možné nastavit, jaké funkce mají být volány při určitých událostech, jako je například `on-click`. Pro výběr dráhy stačí text, který popisuje dráhu a tlačítko, kterým se volá funkce `changeTrack`. Ve výpise 8 je ukázka kódu pro vytvoření dialogového okna, které je na obrázku 14.

```
function showDlg()
   if not ui then
   xml = [[
<ui title="Track Customizer" closeable="true" on-close="hideDlg" resizable="
    false" activate="false">
   <group layout="form" flat="true">
      <label text="Easy track"/>
      <button text="set" on-click="changeTrack" id="1"/>
      <label text="Hill track"/>
      <button text="set" on-click="changeTrack" id="2"/>
      <label text="Intersection track"/>
      <button text="set" on-click="changeTrack" id="3"/>
      <label text="8 track"/>
```

```
      <button text="set" on-click="changeTrack" id="4"/>
      <label text="Complicate track"/>
      <button text="set" on-click="changeTrack" id="5"/>
   </group>
   <label text="" style="* {margin-left: 200px;}"/>
</ui>
]]
      ui=simUI.create(xml)
      if 2==sim.getInt32Parameter(sim.intparam_platform) then
         -- To fix a Qt bug on Linux
         sim.auxFunc('activateMainWindow')
      end
   end
end
```

Výpis 8: Ukázka vytváření uživatelského rozhraní v programu V-REP



Obrázek 14: Výběr přednastavené dráhy

Ve funkci `changeTrack` se uloží do proměnné `trackID` identifikátor tlačítka. Funkce `hideDlg` slouží k zavření okna. Tato funkce je volána pokud není vybrán objekt *MapGenerator*, při zapnutí simulace nebo čištění systémovými funkcemi `sysCall_beforeSimulation` a `sysCall_cleanup`. Skript obsahuje také funkci `getTrackID`, která vrací hodnotu trackID u které je potřeba, aby byla dostupná ze skriptu pro sestavení dráhy.

Kód pro sestavení dráhy se nachází v *child scriptu*. Sestavení dráhy se provádí při inicializaci, tedy při startu simulace. Proto je sestavení dráhy umístěno v systémové funkci `sysCall_init`. Na začátku skriptu jsou uloženy všechny identifikátory jednotlivých dílů dráhy. Také je vytvořeno pole pro pozici křižovatek a proměnná s jejich počtem. Pro polohování dráhy jsou vytvořeny proměnné `x` značící polohu v ose x a `y` značící polohu v ose y. Proměnná `dir` značí směr natočení dráhy.

V další části navazuje kód pro výběr dráhy zvolené přes uživatelské rozhraní popisované výše. Pro zjištění identifikátoru dráhy je potřeba zavolat funkci, která se nachází v *customization scriptu*, `getTrackID`. Podle `trackID` je pak zvolen řetězec reprezentující dráhu. Hlavní část sestavení se provádí v cyklu, který prochází řetězec a podle písmen volá funkce, které připojí daný díl dráhy. Kód pro průchod řetězce je ve výpise 9.

```
for w in string.gmatch(track,"%a+") do
      if w=="S" then buildStraight()
      elseif w=="L" then buildTurn(1)
      elseif w=="R" then buildTurn(-1)
      elseif w=="H" then buildHill()
      elseif w=="U" then buildHillUp()
      elseif w=="D" then buildHillDown()
      elseif w=="I" then buildIntersection()
      elseif w=="O" then buildOffset()
      elseif w=="C" then buildChicane()
      end
end
```

Výpis 9: Cyklus průchodu řetězcem dráhy

Sestavení dráhy se provádí tak, že se podle natočení směru dráhy a délky posune pozice proměnných `x,y` tak, aby odpovídala středu nového prvku dráhy. Na této pozici je volána funkce `buildCopy`, do které je potřeba vložit identifikátor části dráhy, pozici a natočení. Tato funkce posune danou část dráhy na dané místo, otočí ji a vytvoří její kopii. Tato funkce je ve výpise 10.

```
function buildCopy(this, position, rotation)
   sim.setObjectPosition(this,-1,position)
   sim.setObjectOrientation(this, -1,rotation)
   sim.removeObjectFromSelection(sim.handle_all)
   sim.addObjectToSelection(sim.handle_single,this)
   selected = sim.getObjectSelection()
   sim.copyPasteObjects(selected, 0)
end
```

Výpis 10: Kopírování prvku dráhy

Pak jsou ve funkci pro připojení dílu dráhy ještě posunuty proměnné `x,y` na konec dráhy. V případě zatáčky je změněn i směr dráhy. Ve výpise 11 je ukázka implementování funkce pro připojení roviny. Tato funkce obsahuje základ, který je ve všech funkcích pro pokládání dílů dráhy.

```
function buildStraight()
   checkIntersection()
   shiftInDirection(0.360)
   buildCopy(straightHandle, {x,y,0.001},{0,0,math.pi*0.5*dir})
   shiftInDirection(0.360)
end
```

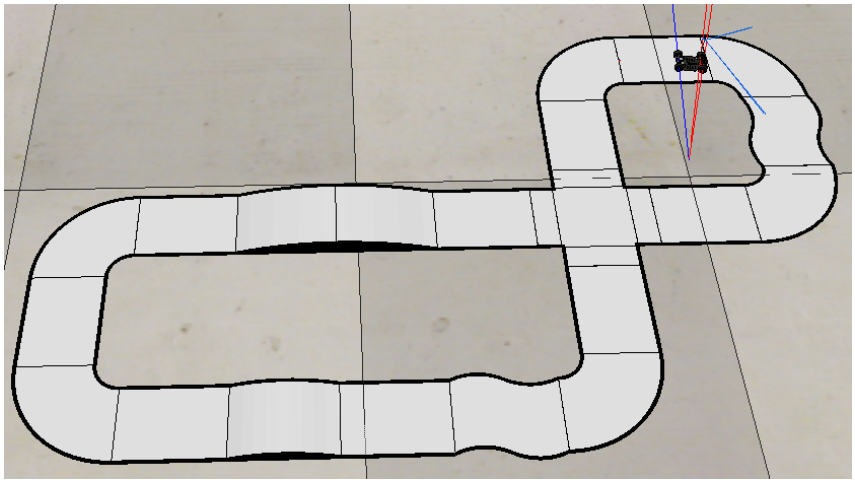
Výpis 11: Posun a položení dílu dráhy

Pro automatické eliminace překrytí křižovatek je ve funkci pro položení křižovatky uložena pozice dané křižovatky a je navýšen počet křižovatek. Při připojování všech částí dráhy se pak kontroluje, zda před sebou dráha nemá křižovatku a pokud ano, tak ji přeskočí posunutím proměnných `x`,`y` ve směru dráhy `dir`. Ke konci skriptu jsou ještě posunuty všechny uložené díly dráhy pod podlahu, aby nebyly vidět ve scéně při simulaci. Nad podlahou tak zůstávají pouze kopie tvořící dráhu.

Jak už bylo řečeno, dráha se sestavuje z řetězce písmen oddělených mezerou. Každý prvek dráhy je představován jedním písmenem. Toto písmeno většinou odpovídá prvnímu písmenu anglického názvu.

- S - straight - rovina.
- L - left curve - levotočivá zatáčka.
- R - right curve - pravotočivá zatáčka.
- H - hill - kopec.
- U - hill up - kopec nahoru.
- D - hill down - kopec dolů.
- I - intersection - křižovatka.
- O - offset - doplnění (krátká rovina).
- C - chicane - šikana.

Na obrázku 15 je možné vidět dráhu odpovídající řetězci: "S R C R S O I O S U D S L S L S H S C L S O O S R". Začátek dráhy je na pozici, na které stojí model auta.



Obrázek 15: Ukázka sestavené dráhy

6 Implementace ovládání pro autonomní řízení

Pro ovládání reálného modelu auta byl v minulosti vytvořen program, který zpracovával obraz z řádkové kamery a na základě těchto dat pak počítal natočení kol a rychlost auta, podle situace na dráze, například průjezd zatáčkou. Toto ovládání vzniklo jako bakalářské práce na kterých pracovali Filip Hanslík [6] a Vojtěch Ihn [7].

Jako rozhraní mezi reálným autem a ovládací částí programu vznikla třída TFC. Tato třída obsahuje několik metod, ve kterých je implementováno přímé řízení auta. Tuto třídu bylo potřeba nahradit totožnou třídou, zajišťující ovládání virtuálního modelu.

Z hlavní části programu byly odebrány nepotřebné části pro komunikaci přes ethernet a byl přidán vstupní parametr programu, kterým je port pro připojení k V-REP rozhraní a volání funkce pro navázání spojení se serverem. Cyklus hlavního programu byl upraven tak, aby se při ukončení simulace ukončil a bylo přidáno zobrazování obrazu z kamery. Dále už byly změny prováděny jen v rozhraní TFC.

6.1 Implementace třídy TFC

Původní rozhraní obsahovalo několik metod pro ovládání LED diod, nebo čtení ze senzorů a tlačítek, které virtuální model nemá a proto bylo hodně metod vypuštěno. Většinou to byly metody zaměřené k testování na modelu.

Oproti ovládání přes ovladač bylo ještě potřeba snížit rychlost auta, aby ho program zvládal ovládat. To bylo provedeno snížením maximálního točivého momentu.

Část programu pro ovládání k výpočtu používala i hodnotu z potenciometru. Aby nebylo potřeba výrazně zasahovat do programu pro ovládání, byly tyto metody nahrazeny metodami, které vrací statickou maximální hodnotu původního rozsahu. Implementace těchto metod je ukázána ve výpise 12.

```
int32_t TFC::ReadPot_i(int32_t channel) {return 1000;};
float TFC::ReadPot_f(int32_t channel) {return 1.0;};
```

Výpis 12: Implementace metod pro čtení potenciometru

Pro připojení k serveru byla přidána metoda `connect`, která přijímá argument, ve kterém předpokládá port, na kterém běží V-REP remote API server. V této metodě probíhá připojení k serveru a také se provádí nutná inicializace, jako je výpočet kroku servomotoru a točivého momentu, získání identifikátorů motorů, serva a senzoru, a započetí streamování obrazu z kamery. Metoda vrací hodnotu 1 pokud se podaří připojit k serveru a 0 pokud se to nepodaří. Implementace této metody je ve výpise 13.

```
int TFC::connect(int portNb)
```

```
{
    clientID=simxStart((simxChar*)"127.0.0.1",portNb,true,true,2000,5);

    if (clientID!=-1)
   {
        stepAngle= MAX_ANGLE_RAD/TFC_SERVO_MINMAX;
        stepTorque= MAX_TORQUE/TFC_PWM_MINMAX;

        simxGetObjectHandle(clientID,"Motor_Left",&leftMotorHandle,
            simx_opmode_blocking);
        simxGetObjectHandle(clientID,"Motor_Right",&rightMotorHandle,
            simx_opmode_blocking);
        simxGetObjectHandle(clientID,"Servo",&servoHandle, simx_opmode_blocking
            );
        simxGetObjectHandle(clientID,"Vision_Sensor",&visionSensorHandle,
            simx_opmode_blocking);

        simxUChar* image;
        int resolution[2];

        simxGetVisionSensorImage(clientID,visionSensorHandle,resolution,&image
            ,1,simx_opmode_streaming);
    return 1;
    }
    else return 0;
};
```

Výpis 13: Metoda connect

Metoda pro ovládání servomotoru jen přepočítává hodnotu na úhel natočení. Tato metoda má jako vstup dva parametry. První parametr určuje, které servo bude ovládáno. To je pozůstatek z původního ovládání auta. Druhý parametr je směr natočení v rozmezí od -1000 do 1000. Tuto hodnotu je potřeba přepočítat na úhel natočení v radianech. Viz výpis 14.

```
void TFC::setServo_i(int32_t channel, int32_t position)
{
   simxSetJointPosition(clientID,servoHandle,position*stepAngle,
      simx_opmode_oneshot);
};
```

Výpis 14: Metoda setServo_i

Metoda pro ovládání motorů převádí hodnotu na točivý moment. Podle směru jízdy také volí rychlost kola v kladné nebo záporné hodnotě. První parametr `pwm_a` je pro ovládání levého motoru a druhý parametr `pwm_b` je pro ovládání pravého motoru. Oba jsou v rozmezí od -1000 do 1000, kde kladné hodnoty jsou pro pohyb vpřed a záporné hodnoty jsou pro pohyb vzad. Implementace této metody je vidět ve výpise 15.

```
void TFC::setMotorPWM_i(int32_t pwm_a, int32_t pwm_b)
{
    int speed_left;
    int speed_right;
    if (pwm_a>0) speed_left = MAX_SPEED_DEG_S; else speed_left=-MAX_SPEED_DEG_S
        ;
    if (pwm_b>0) speed_right = MAX_SPEED_DEG_S; else speed_right=-
        MAX_SPEED_DEG_S;
    if (pwm_a<0) pwm_a=-pwm_a;
    if (pwm_b<0) pwm_b=-pwm_b;
    simxSetJointTargetVelocity(clientID,leftMotorHandle,speed_left,
        simx_opmode_oneshot);
    simxSetJointForce(clientID,leftMotorHandle,pwm_a*stepTorque,
        simx_opmode_oneshot);

    simxSetJointTargetVelocity(clientID,rightMotorHandle,speed_left,
        simx_opmode_oneshot);
    simxSetJointForce(clientID,rightMotorHandle,pwm_b*stepTorque,
        simx_opmode_oneshot);
};
```

Výpis 15: Metoda setMotorPWM_i

Metoda getImage přečte hodnotu senzoru a zapíše ji do vložené proměnné img. Po přepsání obrazu je vymazán buffer. Implementace je ve výpise 16.

```
void TFC::getImage(uint32_t channel, uint16_t *img, uint32_t length)
{
   simxUChar* image_camera;
   int resolution[2];

   int retVal;
   while((retVal=simxGetVisionSensorImage(clientID,visionSensorHandle,
       resolution,&image_camera,1,simx_opmode_buffer)!=simx_return_ok)&&(
       simxGetConnectionId(clientID)!=-1))
   {
```

```
        extApi_sleepMs(1);
    }

    if (retVal!=-1)
    {
        for (int i=0; i<TFC_CAMERA_LINE_LENGTH;i++)
        {
            img[i]= image_camera[i];
        }
    }

    simxGetVisionSensorImage(clientID,visionSensorHandle,resolution,&
        image_camera,1,simx_opmode_remove); //buffer remove
};
```

Výpis 16: Metoda getImage

Součástí TFC je také meteda `finish`, která slouží ke korektnímu ukončení spojení se serverem.

6.2 Výsledky testování autonomního řízení

Model auta se snažil držet uprostřed dráhy. Průjezd zatáčkou 90° model auta zvládal v případě, že nejel vysokou rychlostí. Na rovině model neustále zrychloval a v případě, že byla rovina hodně dlouhá, tak nedokázal projet následující zatáčkou. Zatáčka 180° , neboli dvě zatáčky za sebou, byly pro auto velký problém. Při testování jimi auto nedokázalo projet ani při nízké rychlosti. To může být způsobováno tím, že dané autonomní řízení bylo řešeno pro jinou šířku dráhy a jiný poloměr zatáček. Průjezd šikanou model auta zvládal dobře, stejně jako průjezd křižovatkou.

S kopci měl model auta problém. Ty model dokázal vyjet jen v případě, že před kopcem bylo několik rovin na rozjezd a pokud za kopcem byla zatáčka, tak ji nedokázal projet.

Asi největší problém byl s rychlostí. Autonomní řízení nemá údaje o současné rychlosti. Rychlost je ovládána točivým momentem, který ale nijak neodpovídá rychlosti modelu auta. I při nízkém točivém momentu se na dlouhé dráze může auto jednoduše dostat do neovládatelné rychlosti. Proto by bylo ideální sledovat rychlost motoru nebo kol a na základě toho následně upravovat točivý moment. S tímto vylepšením by se pak mohlo zvýšit i nastavené maximum točivého momentu.

7 Závěr

Cílem mé bakalářské práce bylo vytvořit simulátor autonomního řízení auta v prostředí V-REP. Mým prvním úkolem bylo vytvořit virtuální model odpovídající reálnému modelu auta.

Pro tuto práci byl vybrán model auta Alamak, který se používá při soutěžích autonomního řízení. Na základě tohoto modelu jsem v simulátoru V-REP vytvořil dynamický model a přidal na něj řádkovou kameru. Pro tento dynamický model jsem také vymodeloval vizuální vzhled podle modelu auta Alamak, kterým jsem překryl dynamický model v programu V-REP.

Pro ovládání auta v simulátoru V-REP jsem naprogramoval ovládání z externí aplikace pomocí rozhraní V-REP remote API v jazyce C/C++. Pro optimalizaci ovládání auta byl použit bezdrátový gamepad. Z toho jsem pak vytvořil rozhraní pro aplikaci na autonomní řízení auta, které bylo vytvořené pro reálný model.

Pro testování autonomního řízení bylo potřeba sestavit dráhu. Proto jsem vymodeloval různé prvky dráhy, a pro sestavení dráhy jsem vytvořil skript, který podle zadaného řetězce písmen odpovídajících dílům dráhy, sestavil celou dráhu. Nakonec jsem vytvořil několik tratí, na kterých jsem testoval funkčnost autonomního řízení.

Je těžké odhadnout, jak moc simulované podmínky odpovídají reálným podmínkám. Například obraz z řádkové kamery virtuálního modelu je mnohem čistější, než obraz skutečného modelu. Autonomní řízení dokázalo projet tratí a vypořádat se se všemi prvky dráhy, ale jen v případě, že problémové části byly ideálně umístěné. To je ale problém řešení daného autonomního řízení.

Literatura

[1] V-REP - Virtual Robot Experimentation Platform [online]. [cit. 2019-04-21]. Dostupné z: http://www.coppeliarobotics.com

[2] LANDZO TFC Car model kit [online]. [cit. 2019-04-21]. Dostupné z: http://www.landzo.com/index.php?route=product/product&product_id=95

[3] OpenSCAD: program pro 3D modelování [online]. [cit. 2019-04-21]. Dostupné z: https://www.openscad.org

[4] OpenCV: knihovna pro vykreslování [online]. [cit. 2019-04-21]. Dostupné z: https://docs.opencv.org/4.0.0/

[5] Ackermannova podmínka [online]. [cit. 2019-04-21]. Dostupné z: http://www.autolexicon.net/cs/articles/ackermannova-podminka/

[6] HANSLÍK, Filip. Autonomní řízení auta: optimalizace řízení rychlosti [online]. Ostrava, 2018 [cit. 2019-04-21]. Dostupné z: http://hdl.handle.net/10084/128641. Bakalářská práce. Vysoká škola báňská - Technická univerzita Ostrava. Vedoucí práce Ing. Petr Olivka, Ph.D.

[7] IHN, Vojtěch. Autonomní řízení auta: optimalizace detekce dráhy [online]. Ostrava, 2018 [cit. 2019-04-25]. Dostupné z: http://hdl.handle.net/10084/128642. Bakalářská práce. Vysoká škola báňská - Technická univerzita Ostrava. Vedoucí práce Ing. Petr Olivka, Ph.D.

[8] NXP Cup track configuration [online]. [cit. 2019-04-21]. Dostupné z: https://community.nxp.com/docs/DOC-101612

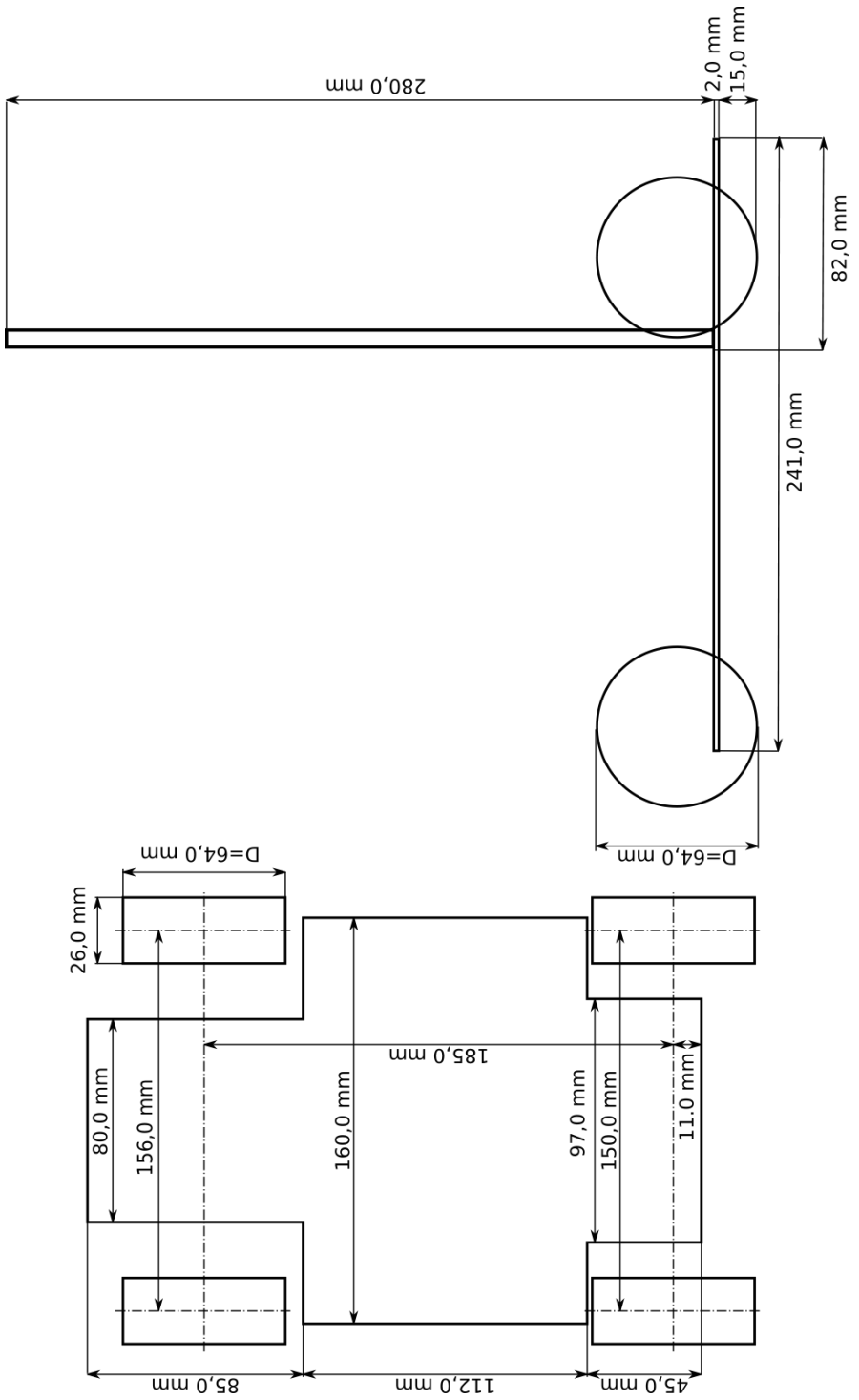
[9] Programming with Joystick on Linux [online]. [cit. 2019-04-22]. Dostupné z: https://kusemanohar.wordpress.com/2015/12/19/programming-with-joystick-on-linux/

[10] Matematické, fyzikální a chemické tabulky a vzorce pro střední školy. Dotisk 1. vydání. Praha: Prometheus, 2003, s. 221. ISBN 978-80-7196-264-9.

A Obsah elektronické přílohy

- 2019_VAS0166_BP_příloha.zip
  - programy
    - autonomni_ovladani
      - CarControl.cpp
      - CarControl.h
      - main.cpp
      - Makefile
      - ProcessedLines.cpp
      - ProcessedLines.h
      - Ring.h
      - tfc.cpp
      - tfc.h
      - bin - obsahuje spustitelný program
      - include
      - remoteAPI
    - ovladacem
      - gamepad.h
      - gamepad_control.cpp
      - Makefile
      - bin - obsahuje spustitelný program
      - gamepad_control
      - include
      - remoteAPI
  - text_bakalarky - zdrojový text BP
    - Figures - použité obrázky
    - 2019_VAS0166_BP.pdf
  - track_models - obsahuje modely dráhy (OpenSCAD)
  - visual_car_models - obsahuje modely částí modelu auta (OpenSCAD)
  - vrep_scena
    - car.ttt - V-REP scéna
  - literatura
    - HAN0221_FEI_B2647_2612R025_2018.pdf
    - IHN0012_FEI_B2647_2612R025_2018.pdf
    - NXP CUP_track_configurations_2018_19.pdf
  - vrep_scripty - obsahuje scripty použité v programu V-REP

B Naměřené rozměry reálného modelu auta



Obrázek 16: Rozměry modelu

C Kódy scriptů

```
// Threaded Child script V-REP pro start remote API serveru
// Asociován s objektem RemoteAPI_Start


function sysCall_threadmain()

   -- Choose a port that is probably not used (try to always use a similar code
      ):
   sim.setThreadAutomaticSwitch(false)
   local portNb=sim.getInt32Parameter(sim.intparam_server_port_next)
   local portStart=sim.getInt32Parameter(sim.intparam_server_port_start)
   local portRange=sim.getInt32Parameter(sim.intparam_server_port_range)
   local newPortNb=portNb+1
   if (newPortNb>=portStart+portRange) then
      newPortNb=portStart
   end
   sim.setInt32Parameter(sim.intparam_server_port_next,newPortNb)
   sim.setThreadAutomaticSwitch(true)

   -- Check what OS we are using:
   platf=sim.getInt32Parameter(sim.intparam_platform)
   if (platf==0) then
      pluginFile='v_repExtRemoteApi.dll'
   end
   if (platf==1) then
      pluginFile='libv_repExtRemoteApi.dylib'
   end
   if (platf==2) then
      pluginFile='libv_repExtRemoteApi.so'
   end

   -- Check if the required remote Api plugin is there:
   moduleName=0
   moduleVersion=0
   index=0
   pluginNotFound=true
   while moduleName do
```

```
        moduleName,moduleVersion=sim.getModuleName(index)
        if (moduleName=='RemoteApi') then
            pluginNotFound=false
        end
        index=index+1
    end

    if (pluginNotFound) then
        -- Plugin was not found
        sim.displayDialog('Error',"Remote Api plugin was not found. ('"..
            pluginFile.."')&&nSimulation will not run properly",sim.dlgstyle_ok,
            true,nil,{0.8,0,0,0,0,0},{0.5,0,0,1,1,1})
    else
        -- Ok, we found the plugin.
        -- We first start the remote Api server service (this requires the
            v_repExtRemoteApi plugin):
        simRemoteApi.start(portNb) -- this server function will automatically
            close again at simulation end
        print("Remote API start at port: "..portNb)

        -- Now we start the client application:
        --result=sim.launchExecutable('/home/naxi/Desktop/bakalarka/ovladacem/bin
            /gamepad_control',portNb,0) -- set the last argument to 1 to see the
            console of the launched client

        result=sim.launchExecutable('/home/naxi/Desktop/bakalarka/
            autonomni_ovladani/bin/projekt',portNb,0)

        if (result==-1) then
            -- The executable could not be launched!
            sim.displayDialog('Error',"Client could not be launched. &&nSimulation
                will not run properly",sim.dlgstyle_ok,true,nil
                ,{0.8,0,0,0,0,0},{0.5,0,0,1,1,1})
        end
    end

    -- This thread ends here. The bubbleRob will however still be controlled by
    -- the client application via the remote Api mechanism!
end
```

```
// Non-Threaded Child script V-REP pro vygenerování dráhy
// Asociován s objektem MapGenerator


function sysCall_init()
   straightHandle = sim.getObjectHandle('track_straight')
   curveHandle = sim.getObjectHandle('track_curve')
   hillHandle = sim.getObjectHandle('track_hill')
   hill2Handle = sim.getObjectHandle('track_hill2')
   chicaneHandle = sim.getObjectHandle('track_chicane')
   offsetHandle = sim.getObjectHandle('track_offset')
   intersectionHandle = sim.getObjectHandle('track_intersection')

   cross = {};
   crossCount = 0;

   x=0;
   y=0;
   dir = 0;

-- Legenda:
-- S - Straight
-- L - Left
-- R - Right
-- H - Hill
-- U - Hill Up
-- D - Hill Down
-- I - Intersection
-- O - Offset
-- C - Chicane

local trackID = sim.callScriptFunction("getTrackID@MapGenerator",sim.
    scripttype_customizationscript)

track = "S S S R S S S R S S S R S S S R" --default if if fail

if trackID==1 then track = "S R S L S R S R S S S O R S S O S R" --easy track
elseif trackID==2 then track = "S C S R S S R S H S R S S R" --hill track
```

```
elseif trackID==3 then track = "S R S R S O I O S L S L S L S O O S R" --
    intersection track
elseif trackID==4 then track = "I O R R R O O L L L O" --8 track
elseif trackID==5 then track = "C S R L S R S S U D C R S R S C L S I O S R S R
     S R S O R S H S R S S H S R S C S" --complicate track
end

--track = "S R C R S O I O S U D C L S L S H C C L S O O S R"
--track = "S R R L L S I O S L L S L O S L R R S S C C S S R R S L S L R S R S
    H"
--track = "S S S R S S S R S S S R S S S R" --You can build your track here



   for w in string.gmatch(track,"%a+") do
      if w=="S" then buildStraight()
      elseif w=="L" then buildTurn(1)
      elseif w=="R" then buildTurn(-1)
      elseif w=="H" then buildHill()
      elseif w=="U" then buildHillUp()
      elseif w=="D" then buildHillDown()
      elseif w=="I" then buildIntersection()
      elseif w=="O" then buildOffset()
      elseif w=="C" then buildChicane()
      end
   end

   sim.setObjectPosition(straightHandle,-1,{10,10,-10})
   sim.setObjectPosition(curveHandle,-1,{10,10,-10})
   sim.setObjectPosition(hillHandle,-1,{10,10,-10})
   sim.setObjectPosition(hill2Handle,-1,{10,10,-10})
   sim.setObjectPosition(chicaneHandle,-1,{10,10,-10})
   sim.setObjectPosition(offsetHandle,-1,{10,10,-10})
   sim.setObjectPosition(intersectionHandle,-1,{10,10,-10})

end
```

```
function buildStraight()
   checkIntersection()
   shiftInDirection(0.360)
   buildCopy(straightHandle, {x,y,0.001},{0,0,math.pi*0.5*dir})
   shiftInDirection(0.360)
end


function buildTurn(turn)
   checkIntersection()
   shiftInDirection(0.360)
   shiftToSide(0.085*turn)

   local lool = 0

   if (turn==-1) then lool = 0.5*math.pi end

   buildCopy(curveHandle, {x,y,0.001},{0,0,(math.pi*0.5*dir) + lool})

   dir = dir + turn;
   if dir==-1 then dir = 3
   elseif dir==4 then dir = 0 end


   shiftInDirection(0.360)
   shiftToSide(0.085*(-turn))
end

function buildHill()
   checkIntersection()
   shiftInDirection(0.360)
   buildCopy(hillHandle, {x,y,0.03},{0,0,math.pi*0.5*dir})
   shiftInDirection(0.360)
end

function buildHillUp()
   checkIntersection()
   shiftInDirection(0.360)
   buildCopy(hill2Handle, {x,y,0.049991},{0,0,math.pi*0.5*dir+(math.pi)})
```

```
    shiftInDirection(0.360)
end

function buildHillDown()
    checkIntersection()
    shiftInDirection(0.360)
    buildCopy(hill2Handle, {x,y,0.049991},{0,0,math.pi*0.5*dir})
    shiftInDirection(0.360)
end

function buildIntersection()
    checkIntersection()
    shiftInDirection(0.275)

    cross[crossCount] = {}
    cross[crossCount][0] = x;
    cross[crossCount][1] = y;
    crossCount = crossCount+1;

    buildCopy(intersectionHandle, {x,y,0.002},{0,0,0})
    shiftInDirection(0.275)
end

function buildOffset()
    checkIntersection()
    shiftInDirection(0.085)
    buildCopy(offsetHandle, {x,y,0.002},{0,0,0.5*math.pi*dir})
    shiftInDirection(0.085)
end

function buildChicane()
    checkIntersection()
    shiftInDirection(0.360)
    buildCopy(chicaneHandle, {x,y,0.002},{0,0,0.5*math.pi*dir})
    shiftInDirection(0.360)
end

function buildCopy(this, position, rotation)
    sim.setObjectPosition(this,-1,position)
```

```
   sim.setObjectOrientation(this, -1,rotation)
   sim.removeObjectFromSelection(sim.handle_all)
   sim.addObjectToSelection(sim.handle_single,this)
   selected = sim.getObjectSelection()
   sim.copyPasteObjects(selected, 0)
end

function shiftInDirection(distance)
   if dir == 0 then x = x + distance
   elseif dir == 1 then y = y + distance
   elseif dir == 2 then x = x - distance
   elseif dir == 3 then y = y - distance
   end
end

function shiftToSide(distance)
   if dir == 0 then y = y + distance
   elseif dir == 1 then x = x - distance
   elseif dir == 2 then y = y - distance
   elseif dir == 3 then x = x + distance
   end
end


function checkIntersection()
   shiftInDirection(0.275)

   for i = 0,crossCount-1 do
      if cross[i][0] - x<0.001 and cross[i][0] - x>-0.001 and cross[i][1]- y
          <0.001 and cross[i][1] - y>-0.001 then shiftInDirection(0.550) end
   end

   shiftInDirection(-0.275)
end
```

```
// Customization Script V-REP pro vytváření dialogového okna pro výběr dráhy
// Asociován s objektem MapGenerator


function sysCall_init()
   objekt=sim.getObjectAssociatedWithScript(sim.handle_self)
   openable = true;
   trackID = 1;
end

function changeTrack(ui,id)
   trackID = id
end

function getTrackID()
   return trackID
end

function showDlg()
   if not ui then
   xml = [[
<ui title="Track Customizer" closeable="true" on-close="hideDlg" resizable="
    false" activate="false">
   <group layout="form" flat="true">
      <label text="Easy track"/>
      <button text="set" on-click="changeTrack" id="1"/>
      <label text="Hill track"/>
      <button text="set" on-click="changeTrack" id="2"/>
      <label text="Intersection track"/>
      <button text="set" on-click="changeTrack" id="3"/>
      <label text="8 track"/>
      <button text="set" on-click="changeTrack" id="4"/>
      <label text="Complicate track"/>
      <button text="set" on-click="changeTrack" id="5"/>
   </group>
   <label text="" style="* {margin-left: 200px;}"/>
</ui>
]]
```

```
      ui=simUI.create(xml)
      if 2==sim.getInt32Parameter(sim.intparam_platform) then
         -- To fix a Qt bug on Linux
         sim.auxFunc('activateMainWindow')
      end
   end
end

function hideDlg()
   if ui then
      simUI.destroy(ui)
      ui=nil
   end
end


function sysCall_nonSimulation()
   local s=sim.getObjectSelection()
   if s and #s>=1 and s[1]==objekt then
      if openable then
         openable = false
         showDlg()
      end
   else
      openable = true
      hideDlg()
   end
end

function sysCall_beforeSimulation()
   hideDlg()
end

function sysCall_cleanup()
   hideDlg()
end
```

```
// Non-Threaded Child script V-REP pro zajišťování možnosti restartu auta
// Asociován s objektem Board


function sysCall_init()
   board=sim.getObjectHandle('Board')
   startPosition = sim.getObjectPosition(board, -1)
   startOrientation = sim.getObjectOrientation(board, -1)
end



function restart()

   handlers = sim.getObjectsInTree(board, sim.handle_all, 2)
   robotInitialConfig = sim.getConfigurationTree(board)
   --sim.setThreadAutomaticSwitch(false)

   for i=1,#handlers,1 do
      sim.resetDynamicObject(handlers[i])
      print(handlers[i])
   end
   sim.setConfigurationTree(robotInitialConfig)
   sim.setObjectPosition(board, -1, startPosition)
   sim.setObjectOrientation(board, -1, startOrientation)
   --sim.setThreadAutomaticSwitch(true)
   print("hello")

   return {},{},{},''
end
```

```
// Non-Threaded Child script V-REP zajišťování Ackermannovy podmínky
// Asociován s objektem Servo

function sysCall_init()
   servo=sim.getObjectHandle('Servo')
   joint_L=sim.getObjectHandle('Joint_Left')
   joint_R=sim.getObjectHandle('Joint_Right')

   l = 0.185
   d = 0.078
end

function sysCall_actuation()

   angle_middle = sim.getJointPosition(servo)

   angle_right=math.atan(l/(-d+l/math.tan(angle_middle)))
   angle_left=math.atan(l/(d+l/math.tan(angle_middle)))

   --print(angle_left.." "..angle_middle.." "..angle_right)
   --print(((angle_left/math.pi)*180).." "..((angle_middle/math.pi)*180).." "
       ..((angle_right/math.pi)*180))


   sim.setJointPosition(joint_R, angle_right)
   sim.setJointPosition(joint_L, angle_left)
end
```

```
// Non-Threaded Child script V-REP pro vykreslování osy kola
// Asociován s LineSegment (dummy objekty na kolech)


function sysCall_init()
   h=sim.getObjectAssociatedWithScript(sim.handle_self)
   dh=sim.addDrawingObject(sim.drawing_lines,1,0,h,2,nil,nil,nil,{0,0,1})
   m=sim.getObjectMatrix(h,-1)
   --pt1={0,0,0.25}
```

```
    --pt2={0,0,-0.1}
    pt1={0,0,1}
    pt2={0,0,-1}
    pt1=sim.multiplyVector(m,pt1)
    pt2=sim.multiplyVector(m,pt2)
    lineData={pt1[1],pt1[2],pt1[3],pt2[1],pt2[2],pt2[3]}
    sim.addDrawingObjectItem(dh,lineData)
end

function sysCall_cleanup()
    sim.removeDrawingObject(dh)
end
```